



KNEECAP

model-based generation
of network traffic

<http://github.com/niksu/kneecap>

Nik Sultana
SMT2016, 2nd July

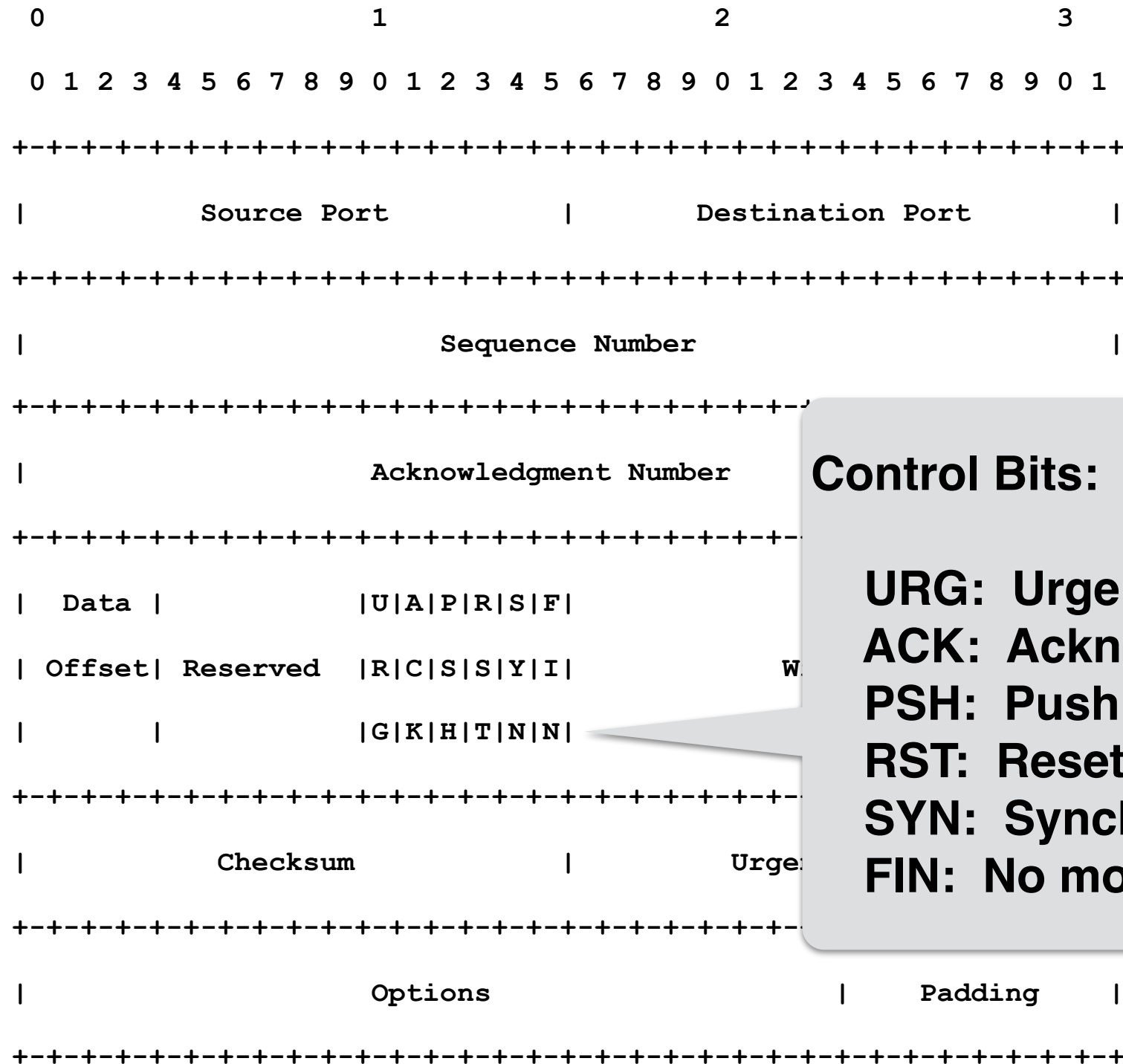
Why craft packets

To evaluate:

- Behaviour (and security)
- Performance

of software, hardware, and their configuration

TCP Header Format



Control Bits: 6 bits (from left to right):

- URG:** Urgent Pointer field significant
- ACK:** Acknowledgment field significant
- PSH:** Push Function
- RST:** Reset the connection
- SYN:** Synchronize sequence numbers
- FIN:** No more data from sender

from <https://tools.ietf.org/html/rfc793>

For example

```
from scapy.all import *
from random import randint
# Create the skeleton of our packet
template = IP(dst="172.16.20.10")/TCP()
# Start lighting up those bits!
template[TCP].flags = "UFP"
# Create a list with a large number of packets to send
# Each packet will have a random TCP dest port for attack obfuscation
xmas = []
for pktNum in range(0,100):
    xmas.extend(template)
    xmas[pktNum][TCP].dport = randint(1,65535)
# Send the list of packets
send(xmas)
```

<https://thepacketgeek.com/scapy-p-08-making-a-christmas-tree-packet/>

https://en.wikipedia.org/wiki/Christmas_tree_packet

```
from scapy.all import *
from random import randint
# Create the skeleton of our packet
template = IP(dst="172.16.20.10")/TCP()
# Start lighting up those bits!
template[TCP].flags = "UFP"
# Create a list with a large number of packets to send
# Each packet will have a random TCP dest port for attack obfuscation
xmas = []
for pktNum in range(0,100):
    xmas.extend(template)
    xmas[pktNum][TCP].dport = randint(1,65535)
# Send the list of packets
send(xmas)
```

Imperative style:

- **assignment**
- **use “For” loops for parameter sweeping.**

Example **goal** spec

```
<@ tcp.URG &&  
    tcp.PSH &&  
    tcp.FIN @>
```

Goal

Declarative style: use formulas

~~Imperative style:~~

- ~~• assignment~~
- ~~• use “For” loops for parameter sweeping.~~

Why make it **declarative**?

- Improved readability, **convenience**.
- Better **compositionality** of specs.
- Explore opportunity for better **automated support**.
- Explore use of tools to provide such support.

Packets as formulas

01 01 00 00 00 20 64 00 19 50 10 04 55 48 02 07
08 00 00 00 60 00 80 00 01 50 20 00 02 00 00 29
00 60 00 80 80 04 20 23 00 11 00 0d 85 42 00 10
24 00 41 02 04 00 08 44 02 42 00 02 50 40 00 04
08 00 64 10 0c 81 20 40 01 04 00 20 00 20 c4 04
08 30 05 80 04 02 04 02 21 40 00 40 04 00 20 11
00 10 80 00 0a 00 02 00 00 00 10 24 00 00 00 80
34 08 10 84 a2 a2 04 00 d0 01 00 86 38 40 02 80
00 01 47 00 00 10 08 00 00 40 15 04 41 00 10 02

$$= V_1 \cdot V_2 \cdot V_3 \cdot V_4 \cdot V_5 \cdots V_n$$

$$\& V_1 = 01\ 01\ 00\ 00\ 00\ 20$$

$$\& V_2 = 64\ 00\ 19\ 50\ 10\ 04$$

$$\& V_3 = 55\ 48\ 02\ 07$$

...

$$\text{where } |V_1| = 48$$

$$|V_2| = 48$$

$$|V_3| = 32$$

...

Packet **formats** as formulas

```
01 01 00 00 00 20 64 00 19 50 10 04 55 48 02 07
08 00 00 00 60 00 80 00 01 50 20 00 02 00 00 29
00 60 00 80 80 04 20 23 00 11 00 0d 85 42 00 10
24 00 41 02 04 00 08 44 02 42 00 02 50 40 00 04
08 00 64 10 0c 81 20 40 01 04 00 20 00 20 c4 04
08 30 05 80 04 02 04 02 21 40 00 40 04 00 20 11
00 10 80 00 0a 00 02 00 00 00 10 24 00 00 00 80
34 08 10 84 a2 a2 04 00 d0 01 00 86 38 40 02 80
00 01 47 00 00 10 08 00 00 40 15 04 41 00 10 02
```

$\in V_1 \cdot V_2 \cdot V_3 \cdot V_4 \cdot V_5 \cdots V_n$

& $V_1 \in P_1$

...

& $V_n \in P_n$

(Symbolic) **Packet**

$$\left(\begin{array}{l} \text{pckt} = V_1 \cdot V_2 \cdot V_3 \cdot V_4 \cdot V_5 \cdots V_n \\ \& V_1 = 01\ 01\ 00\ 00\ 00\ 20 \\ \& V_2 = 64\ 00\ 19\ 50\ 10\ 04 \\ \& V_3 = 55\ 48\ 02\ 07 \\ \dots \end{array} \right)$$

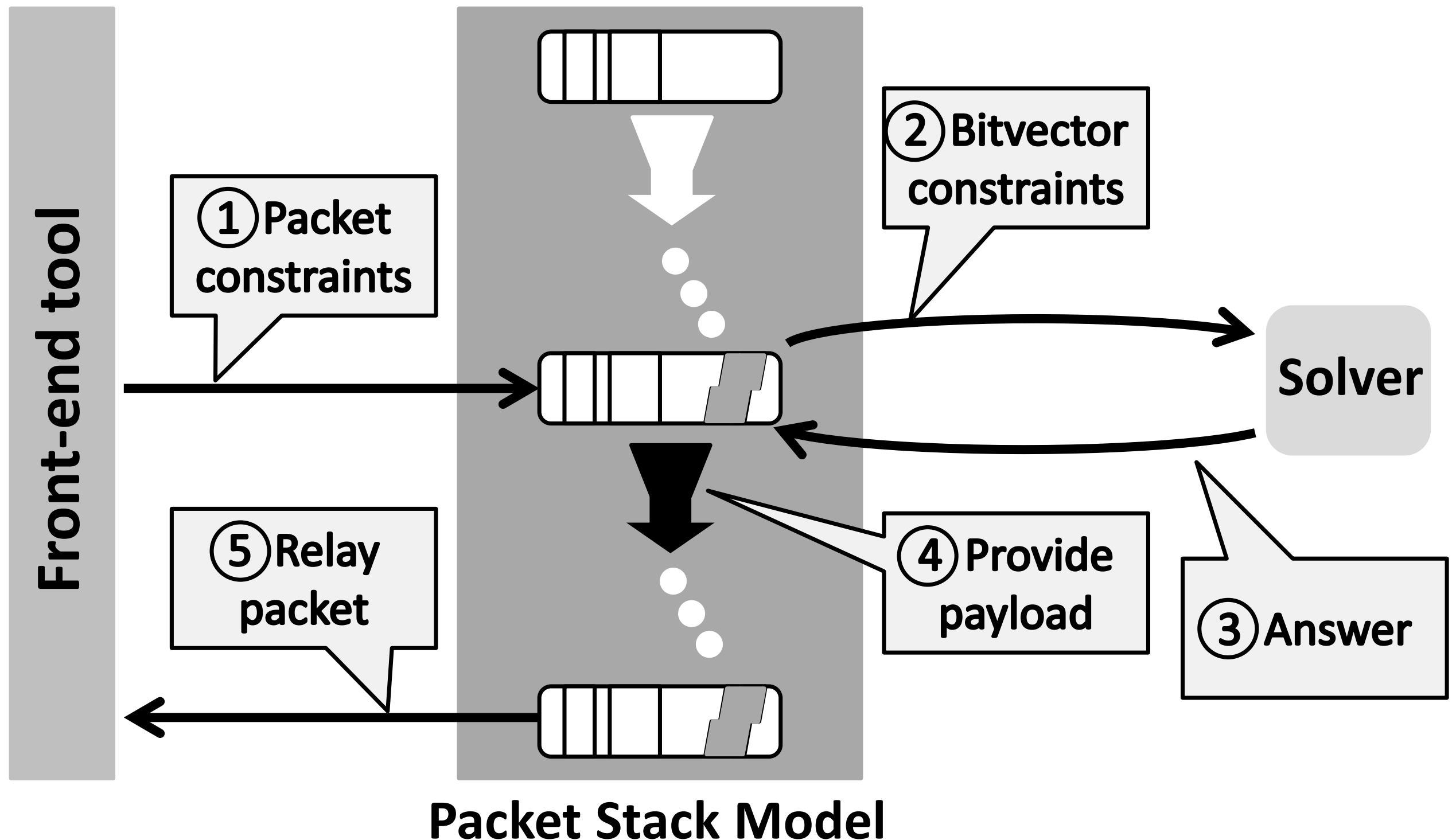
Packet **Format**

$$\left(\begin{array}{l} \text{pckt} = V_1 \cdot V_2 \cdot V_3 \cdot V_4 \cdot V_5 \cdots V_n \\ \& P_1(V_1) \\ \& \dots \\ \& P_n(V_n) \end{array} \right)$$

Encapsulation



Architecture



Translation

```
ipv4.version = 4 &&  
ipv4.source_address = ipv4.ipv4_address "10.10.10.[55-60]" &&  
ipv4.source_address = ipv4.destination_address &&  
ipv4.internet_header_length = 5 &&  
ipv4.total_length = 170 &&  
ipv4.TTL = 5 &&  
ipv4.protocol = ipv4.protocol_ip_in_ip
```

Translation

```
ipv4.version = 4 &&  
ipv4.source_address = ipv4.ipv4_address "10.10.10.[55-60]" &&  
ipv4.source_address = ipv4.destination_address &&  
ipv4.internet_header_length = 5 &&  
ipv4.total_length = 170 &&  
ipv4.TTL = 5 &&  
ipv4.protocol = ipv4.protocol_ip_in_ip
```

Distinguished
constant

Interpreted
constant

Literal

```
ipv4.version = 4 &&
```

```
ipv4.source_address = ipv4.ipv4_address "10.10.10.[55-60]" &&  
ipv4.source_address = ipv4.destination_address &&  
ipv4.internet_header_length = 5 &&  
ipv4.total_length = 170 &&  
ipv4.TTL = 5 &&  
ipv4.protocol = ipv4.protocol_ip_in_ip
```

```
ipv4.version = 4 &&  
ipv4.source_address = ipv4.ipv4_address "10.10.10.[55-60]" &&  
ipv4.source_address = ipv4.destination_address &&  
ipv4.internet_header_length = 5 &&  
ipv4.total_length = 170 &&  
ipv4.TTL = 5 &&
```

```
ipv4.protocol = ipv4.protocol_ip_in_ip
```

Interpreted
to literal

Custom map to
expr+constraint

```
ipv4.version = 4 &&  
ipv4.source_address = ipv4.ipv4_address "10.10.10.[55-60]" &&  
ipv4.source_address = ipv4.destination_address &&  
ipv4.internet_header_length = 5 &&  
ipv4.total_length = 170 &&  
ipv4.TTL = 5 &&  
ipv4.protocol = ipv4.protocol_ip_in_ip
```

```
use eth = new ethernet(184u)
use ip = new ipv4(180u)

eth.constrain <@ ethernet.source_address = ethernet.mac_address "[1-5,10]:34:56:78:90:*"
    && ethernet.ethertype = ethernet.ethertype_ipv4 @>
<== ip.constrain <@ ipv4.version = 4 &&
    ipv4.source_address = ipv4.ipv4_address "10.10.10.[55-60]" &&
    ipv4.source_address = ipv4.destination_address &&
    ipv4.internet_header_length = 5 &&
    ipv4.total_length = 170 &&
    ipv4.TTL = 5 &&
    ipv4.protocol = ipv4.protocol_ip_in_ip
    @>
```

As bitvector formulas: **ethernet**

```
<@ ethernet.source_address = ethernet.mac_address "[1-5,10]:34:56:78:90:*"  
  && ethernet.ethertype = ethernet.ethertype_ipv4 @>
```

```
(let ((a!1 (concat (concat (concat (concat range0 #x34) #x56) #x78) #x90)))  
(let ((a!2 (=> (not (= src_mac (concat a!1 wild1)) false)))  
  (and (=> (= src_mac (concat a!1 wild1)) (= ethertype #x0800))  
    a!2  
    (or (= range0 #x0a)  
      (= range0 #x01)  
      (= range0 #x02)  
      (= range0 #x03)  
      (= range0 #x04)  
      (= range0 #x05))))))
```

Help

How to influence the
distribution of models?

From earlier

```
from scapy.all import *
from random import randint
# Create the skeleton of our packet
template = IP(dst="172.16.20.10")/TCP()
# Start lighting up those bits!
template[TCP].flags = "UFP"
# Create a list with a large number of packets to send
# Each packet will have a random TCP dest port for attack obfuscation
xmas = []
for pktNum in range(0,100):
    xmas.extend(template)
    xmas[pktNum][TCP].dport = randint(1,65535)
# Send the list of packets
send(xmas)
```

Feedback and
pull-requests welcome.

<http://github.com/niksu/kneecap>